Assessing Security Risk In Legacy Systems

Carl Weber, Cigital, Inc. [vita[1]]

2006-12-14

This article outlines a general approach to assessing the security risk posed by your existing business systems#the systems already in place within your organization. This approach has four steps: identifying and describing the legacy systems, selecting the legacy systems that are most likely to present a risk, quickly evaluating the risks that each “most likely” system poses to the enterprise, and finally developing a strategy to mitigate risk, reducing it to an acceptable level.

Intended Audience

Executives concerned about the security of the systems they currently are using will find this article useful. It offers an approach to determine the extent of the security problem that you are facing and describes options available for addressing that problem.

Developers involved in maintaining existing systems will also find this article useful. It outlines some key considerations in mitigating security problems in existing systems and migrating existing systems to new environments.

Software Security in Legacy Systems

The term *legacy system* has different connotations for different people. To many it refers to old mainframe, dumb-terminal applications from the 1970s and 1980s. To others it may imply the client/server systems of the 1990s. To some it may suggest the first generation of web-based business applications developed in the late 1990s. These systems have fundamentally different architectures and present different risks.

Often the dumb-terminal and client/server systems are dismissed as old, rare, and irrelevant. Yet that is emphatically not the case. These systems, or portions of them, are still in widespread use, though they may now have different user interfaces.

Do not assume that these older systems are free from risk because they were built at a time when security was not a design issue, computer crime was rare (or invisible), and the mechanisms for attack were different, generally relying on physical access and inside knowledge. These systems are now in the open, unprotected by the data center, and they are vulnerable.

This article is about securing your existing systems today. For our purposes, a legacy system is any system that is currently in use within your enterprise. Any system that currently is accessible to users and is fulfilling some useful function is a legacy system.

Types of Legacy Systems

For the purposes of assessing security it makes sense to categorize systems by the way users interact with them. Software security involves reacting correctly to inputs into a system and resisting attempts to compromise a system by executing its components in an unintended sequence [McGraw 06[2]]. Your existing legacy systems get their inputs from users, directly through an input screen or indirectly through previously prepared files or databases. Users normally control execution of a system’s components by selecting menu choices.

1. http://buildsecurityin.us-cert.gov/bsi/about_us/authors/640-BSI.html (Weber, Carl C.)
2. #dsy624-BSI_mcgraw06

The way users interact with systems has changed radically over the past forty years or so, from mainframe terminals, to client/server desktop workstations (PCs), to browser-based interfaces on a myriad of stationary and mobile devices. Each method of user interaction raises its own set of software security considerations.

Terminal (Mainframe) Systems

Description

Widespread use of computers first occurred in the 1960s. The technology then was a large central processor#a mainframe in the beginning, with minicomputers entering the market later. Users accessed the central processor with simple, directly attached terminals. Mainframe technology remained the dominant practice through the mid-1980s, and remains today a key element of the data processing environment in many organizations. Many business systems still use central processors and dumb terminals for reasons of operating cost, the expense and risk of replacement, and the speed of the user interface. Car rental, travel agent, and airline gate agent terminals are common examples of devices often supported by mainframe technology. The technology looks clumsy, but it works fast and well in the hands of a skilled user.

Initially, these terminals were physically hard-wired to the central processor. Soon dial-in modem access was supported. Eventually networks connected the central processors together; a user could use a terminal emulator program running on one platform to access an application system running on another. Today, mainframe users often use terminal emulators running on the workstations on their desktop.

The Sabre airline reservations system, originally developed by American Airlines forty years ago, is a classic example of an evolving mainframe system. Sabre originally relied on simple, directly connected terminals located in airports and at American’s reservations centers. Then specialized workstations for travel agents’ desktops were developed. Today users can access Sabre through web browsers running on home computers.

Software Security Considerations

Systems using dumb terminals appear more secure than modern applications. The terminal has little or no processing capability, and user interaction is strictly limited by the application or, for some categories of users, by a very limited command-driven operating system interface. Certainly, these constraints provide some measure of assurance, but the security is not absolute.

Some points to consider when examining a dumb-terminal system for security weaknesses:

- Developers gave software security little if any consideration when the first dumb-terminal systems were developed. The physical isolation provided by hard-wired terminals made it unnecessary to think about unknown attackers attempting to penetrate the system. Physical security ended with the advent of modem access and expanding networks. In general, developers made little effort to examine or remediate security at the application level for the earliest systems despite the changed circumstances.
- Dumb-terminal systems rely on user authentication provided by the central processor’s operating system. The user enters the ubiquitous account number and password, which the central processor authenticates.
- Systems developed for these central processor platforms invoke operating system functions to identify the current user. Access to system resources is then limited based on the logon ID. The most common method to control access is based on file-level protections—the right to execute a program is controlled by limiting access to the executable file. The systems that actually block unauthorized access are essentially bulletproof. The risk is in the human management of these systems, such as making sure that the correct restrictions are in place for a file and controlling passwords. Because mainframe systems share common resources, individuals often have a less proprietary interest in protecting the asset.
- The application system usually provides the authorization logic to limit user access to restricted functions and data. Attackers often exploit weaknesses in the operating system’s login facility to penetrate the platform or find holes in the application system’s authorization logic to get to restricted functions and data.

- Session management is not a problem, per se, in a dumb-terminal system. The connection between a terminal and the central processor represents a session when hard-wired terminals are involved. Sessions are less obvious when the connection is through a modem or network.
- Dumb terminals cannot handle encryption. As a result, unencrypted data is often flowing across the network between the terminal platform and the processing platform.
- Dumb terminals simply accept input from users without local validation. The application code must interpret this raw input stream of characters properly. Developers attempt to write code that gracefully handles anything that the user enters, but, inevitably, they may miss something that would allow an attacker to enter some unexpected value that would cause the system to crash or behave in an incorrect way.
- Software running on modern PCs can easily emulate a dumb terminal. Software designers may have made assumptions in their design based upon the characteristics and behavior of a particular terminal device, but an attacker can easily make the software emulator behave differently. For example, the classic IBM 3270 page-mode terminal provided features that would limit input values in input fields to numeric characters and restrict the length of entered text fields. But an attacker using software emulating a 3270 can compromise any application software that assumes these validations rules have been properly applied. The attacker simply breaks the rules.

Workstation (Client/Server) Systems

Description

In the 1980s personal computers began appearing on user desktops. Soon local area networks (LANs) connected these PCs together. The PCs accessed central facilities like data storage and printing through the LANs.

System developers started building systems that took advantage of this desktop processing capability. These client/server systems actually executed on the PCs, or workstations. Information common to the enterprise was stored in centralized database servers.

Software Security Considerations

Client/server applications are different from and inherently more complicated than systems operating within a single processor. Two distinct and highly capable systems#the client and the server#perform their roles in a complex ballet. Substantial information flows back and forth, and each side relies on the other to perform its function correctly.

In the history of IT, these systems had the most complex and delicate architectures and required the most maintenance. These factors led to the decline of client/server architectures in favor of thin-client systems. But many important systems with client/server architectures remain in use, including hybrid systems that incorporate client/server concepts.

Some points to consider when evaluating security in workstation, client/server systems:

- The server is the focus of security attention. A properly designed client/server system does not permanently retain information on the client (other than configuration data). An attacker gains little by penetrating the client, unless that success facilitates a subsequent attack on the server. Of course, not all clients are designed properly. The temptation to store some private or sensitive data on the client to improve speed of execution is strong. This was particularly true when network speeds were much lower than today. Many client/server systems date from that era.
- To compromise a client/server system, attackers need a network-level connection to the server. Network-level security becomes a major issue; can an attacker outside of the enterprise gain access to the system’s server? Are firewalls in place to prevent this?
- Attackers must access the database on the server. Every legitimate client accessing that database must have the database login ID and password to access it. How is the database login data retained on the

client? Is it encrypted or otherwise protected in some way, and how is it transmitted across the network to the server?

- Attackers can access the database server directly, without the client, using various generic tools. The attacker does not need the client software to penetrate the system. Often having the client software is useful; the attacker can learn much about the system and its data by analyzing, de-compiling, and running the client software. But damage can be done without it.
- Distribution of client software in client/server systems was and is a major problem. When developers change the client software, how does the revised software get installed on the user workstations? If installation isn't easy or automatic, users often don't do it, creating configuration management problems. But arrangements that automate the update process often provide attackers with new opportunities to compromise a client/server application. Attackers can hijack auto-update features so that their compromised version of the client software is actually installed on the client workstation. This compromised version could contain spyware that provides the attacker with information about the user's activity or performs invisible accesses and updates to the central database.
- Older versions of client software, perhaps lacking security protections built into newer versions, can provide opportunities to attackers to compromise the server. The server still trusts the older versions#in the spirit of backward compatibility. An attacker can use an older version of the client to circumvent restrictions included in newer versions. Schemes to improve the security of a client/server system must take the older versions of the client into account. This fact reinforces the point that security in client/server situations focuses on the server; you cannot trust the client.
- A *session* in a client/server system is essentially the connection between the client software and the server database. Establishing this connection is expensive in terms of the resources and time required, so applications typically make the connection when the session starts and then keep it open. An attacker can take over a session by spoofing, making his workstation look like the workstation of a legitimate user who has already established the connection to the database. The database server cannot distinguish between the attacker's workstation and the legitimate user's workstation.
- Interactions with the database are transaction-based, with commit actions to complete a transaction. Proper design requires that transactions are completed quickly and not left open; otherwise, a break in the client/server connection leaves the data in an indeterminate state. When this happens, the database executes a rollback action to undo the partially completed transaction. An attacker can exploit this characteristic by disrupting the client to server connection at critical points, before the commit action has occurred. A well-formed attack of this type could cause transactions that the user thought were properly recorded in the database to be essentially ignored.
- In general, attacks that crash the client do not really accomplish much. The user immediately affected is inconvenienced but the system as a whole continues in operation. The real goal of an attacker is to crash the central database server, which affects everyone.

Browser (Internet) Systems

Description

The arrival of the Internet in the mid-1990s provided a new way to access business systems. Users can perform data access and updating through web pages, invoked through a universally available browser program running on a connected workstation. A central web server generates these web pages in response to a request from a remote browser. Internet access gave any Internet user, anywhere in the world, potential access to your business systems.

Browser-based information processing has evolved considerably over the last ten years, but the principle remains the same. A browser requests a web page from a web server. The web server reacts to request by providing the requested page. The browser's request can contain information that it wants the web server to retain indefinitely; enhancements on the web server look for this additional information in the request and handle it properly.

Browser-based systems are currently the most common systems that most organizations deploy. Many are public facing, meaning any Internet user can access them. Others are private, operating on a (hopefully) isolated network and accessible only to users on that network. Developer attention has concentrated on making these systems easy to use, inexpensive to develop, reliable, high performance#and secure.

Various tools are available to develop the code that processes browser requests on the server. These tools have evolved considerably since the mid-1990s, in terms of their reliability, performance, and ease of use. The current state-of-the-art browser-based systems use a framework-based tool with a strongly typed, object-oriented language such as J2EE/Java or .NET/C#. Other systems rely on tools such as Active Server Pages (ASP), Cold Fusion, PHP, Perl, C/C++, and myriad other languages and products.

Software Security Considerations

Much has been written about security issues related to Internet business systems. These issues will not be repeated here. But realize that even Internet-based systems can be legacy and that all considerations that apply to the security of new systems must be applied to those already in use. Here, we simply contrast the issues in Internet applications to those in client/server and dumb-terminal applications.

- The reassurance provided by the physical connectivity of terminal-based systems, and even client/server systems, is lost completely with browser-based systems. Attackers with Internet access anywhere in the world can access public facing browser systems (and private browser-based systems if the subnetworks they operate on are not properly isolated). As physical isolation is lost it must be replaced by “logical” isolation, which relies on firewall/router equipment. Of course, this equipment is executing software, which in itself may be vulnerable to attack.
- The identity of attackers is relatively easy to hide. Various tricks can render an attacker’s access anonymous. These tricks include operating through anonymous proxy servers and spoofing the identity information provided by client browsers.
- The issues described elsewhere on the Build Security In web site for new systems apply to existing browser-based systems. Authentication/authorization is critical in keeping undesirable users away from restricted functionality or sensitive data. Session management must be handled properly to prevent hijacking sessions and impersonating other users. All inputs must be completely validated to prevent buffer overflows and SQL injection and cross-site scripting. Sensitive data must be encrypted when it is passed across a network.
- Security issues with browser-based systems will vary significantly, depending upon the technology used to implement the system. For example, older system implementations that used C or C++ to code web server routines tend to be vulnerable to potential buffer overflow problems caused by the usage of inappropriate functions. More recent systems based on a solid framework and using a strongly typed language generally avoid these problems, but these systems are susceptible to deficiencies in the implementation of the framework.
- In the late 1990s the arrival of the Internet coincided with concerns about legacy system operations in 2000 and beyond#the “Y2K” crisis. Many organizations hastily constructed Internet-based systems to replace the older systems that posed Y2K concerns. These replacement systems, developed using the relatively primitive tools available then, often were developed without proper attention to security.
- Evolving Internet technologies, such as web services and AJAX, constantly raise new security concerns for browser-based systems. The technologies are intended to improve the user interface experience and reduce software development effort and cost, but they inevitably create opportunities that attackers can exploit to compromise a system. Web services are intended to facilitate connecting users to common functionality, which is a great improvement until someone misuses the capability for nefarious purposes. AJAX provides a more intimate interaction between the browser client and web server, which provides a more dynamic user interface experience but also provides new ways for an attacker to get into the server.

Assessing Legacy System Security Concerns

The first step in solving a security problem is to understand it. This section describes a simple approach to developing a better understanding of the extent of the risk posed by your existing legacy systems to your enterprise.

Security risk associated with individual systems is generally determined by applying the Risk Management Framework[3], or RMF. Briefly, RMF involves performing a series of steps that establish an understanding of the business context and then investigate technical risks and their effect on the goals of the organization. It is a closed-loop process, in that it evaluates the effect of the corrections on reducing the business risks. It involves a detailed, methodical analysis of existing legacy systems.

This section outlines an abbreviated approach to assessment, based loosely on the RMF. This approach will with minimal effort provide a rough sense of the scope of the problem that your legacy systems represent.

Step 1: List All Legacy Systems

Create a list of the systems in your enterprise that are in use today and are accessible by users. This seemingly simple task is always difficult in an enterprise with a long history of IT activity, but you cannot manage what you do not know you have. Old systems often are not formally retired but simply used less and less; eventually they are available for use, but never actually used. Other old systems are used frequently but are generally ignored because they are hidden behind other systems, so that users may not even realize they are using them.

Describe each system as you list it:

- Briefly describe what the system does.
- Indicate when the system was first placed in production.
- Describe the circumstances of its development#in-house custom system, contractor-developed custom system, commercial off-the-shelf (COTS) system, etc.
- Categorize the system by its type, using the types of legacy systems listed above.
- Note facts about its implementation#the language and framework used, the type of database, etc.
- Indicate whether the system is accessible by the public, internal use only, etc.
- Indicate whether the system handles sensitive information.
- Record other relevant facts that might be readily known, such as the availability of documentation, extent of testing that it was subjected to, etc.
- Identify an internal contact person who can provide additional information about the system.

Listing legacy systems would seem to be a trivial task for a well-managed IT organization. Our experience, however, is that few organizations of any size truly know of every system operating in the organization. Old systems, old versions, abandoned and nearly abandoned systems, and systems installed by users and business units without IT involvement litter all major enterprises. Each of these is a potential vulnerability.

Systems inadvertently omitted from this inventory could represent a latent risk to the enterprise of which management is unaware. The quality of the overall assessment depends upon the completeness of this list.

Step 2: Identify Potentially High-Risk Systems

Focus on the legacy systems that deserve further attention from a security perspective. Pinpoint the ones that present the greatest risk to the enterprise. Limit this list to a small subset of all of the existing legacy systems. Ultimate protection, of course, requires that all systems be examined, but this is very expensive, not typically cost effective, and the too-broad focus will delay action on the most important systems.

Do a quick, common-sense assessment of each system on the list. A formal risk assessment involves a time-consuming review of the system’s characteristics, but analysis that continues too long will delay imperative

3. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/risk/250-BSI.html (Risk Management Framework (RMF))

action. Further, no IT organization really needs to derive priorities from first-principles#the organization probably has a good sense already where the biggest problems lie. Certainly, you should question the "conventional wisdom," examining it periodically as time permits. But you can start with the information readily at hand.

Consider these factors:

- Data sensitivity: If you know that the system accesses sensitive data#credit card numbers or social security numbers or pay rates, for example#then the system is likely to present a significant risk if an attacker compromises it.
- Functional criticality: If the system supports business functions that are critical to the fundamental operation of the enterprise#the order entry system for an online retailer, for example#then having an attacker crash it or otherwise compromise it in some fashion will be a major impact on the enterprise.
- Government requirements: If the system maintains information that is subject to government security requirements, the risk to the enterprise is significantly increased. Recent corporate scandals and privacy legislation have created new rules#HIPAA, GLBA, Basel2, Sarbanes-Oxley, etc.#to which systems must conform.
- Accessibility: If the users access the system with a web browser through the Internet, then it probably represents a greater risk than if a PC workstation or terminal is used.
- Internal versus external usage: Publicly accessible systems present different risks than systems only accessible to internal users.
- Age, implementation tools: If the system was recently implemented with J2EE or .NET, then it may pose less risk than a system implemented ten years ago using CGI. Similarly, if it is an old mainframe system that has been running reliably and without compromise for many years, then it probably is okay for another few years.
- The critical caution on the "old reliable" system is that it may have changed over the years without too much high-level attention. Often half of all systems development work occurs after initial deployment [Glass 03[4]], often in small steps with less attention to process and testing than for new systems. Particular attention should be paid to how the oldest systems interface to enterprise critical assets (like the database) and to other systems using new technology. The legacy system, which has historically operated in the safety of the bowels of the mainframe, may be compromised through a simple interface to an interconnected system. The reliable dumb terminal that was assumed when the system was written may have been replaced by terminal emulation software that does not enforce the same rules on input validation.
- Extent of security testing: If developers realized that the system represented a significant potential risk and therefore tested it extensively, you can have more confidence in it compared with other systems that were untested. Remember, security testing performed after a system is developed is not an absolute assurance that the system has no vulnerabilities; it only indicates that the tests performed revealed no vulnerabilities at that point in time.

Keep the number of systems that you select for further review down to a reasonably small number. Each selection requires significant effort to investigate, but do not omit a system simply because little is known about it. If necessary, spend the time needed to learn enough to make a reasonable decision.

Step 3: Assess Potentially High-Risk Systems

Conduct a more complete investigation of each system identified as a potentially high risk. Identify the attack patterns that could damage the enterprise. Determine with reasonable precision the level of risk that these patterns represent to the enterprise, individually and in summary.

This step performs the basic investigative tasks of the Risk Management Framework but at a relatively high level of detail. You must "think like an attacker"—look at the system from the attacker's perspective, understand the attacker's motivations, and consider the attack patterns that an attacker would employ to compromise the system. Reference materials, such as the text "Exploiting Software#How To Break

Code" [Hoglund 04[5]], provide a good background for appreciating the threats that your systems must deal with.

Begin by confirming your assumptions that led you to place this system on the potentially high-risk system list. Talk with the system's contact person#someone familiar with the system, its functionality, its history. Verify that the system handles the data and supports the business functions that you understood it performs. Verify the other attributes of the system#age, implementation tools, user access methods, extent of testing, etc.#that influenced the decision.

Reconsider the decision to include the system, using the verified information. You may find the assumptions that caused this system to be on the list are incorrect. In that case, drop the system from further investigation.

Next, conduct a more thorough review of the system. Talk to the system's stakeholders about potential problems that they foresee. Talk with developers about how they addressed security concerns during development. Talk with operations personnel and network security people to learn about security problems that may appear in logs or events that have occurred. Review system documentation to investigate how it addresses security issues. Examine test results, if available, to verify the extent of security testing that was performed.

Focus this review on two basic questions: what are the business risks to the enterprise that the operation of this system creates, and how effective is the system design in handling those risks. Also, answer a third, related question: how severe is the risk to the enterprise.

Develop a list of the attack patterns that conceivably could be used against the system. Evaluate the likelihood#the feasibility#of executing each pattern, and estimate the impact on the enterprise if the execution of the attack pattern is successful.

Identifying the attack patterns involves considering hypothetical situations: what if an attacker successfully impersonates a legitimate user, what if an attacker gains access to the database, what if an attacker compromises a fundamental business process, etc. The "think like an attacker" mentality is important in conducting this review.

You must know the business purpose of the system to understand the motivations of an attacker, to imagine the possible attack patterns, and to appreciate the implications of a successful attack. Would a successful attack that accesses the database be a mere inconvenience to the enterprise or threaten its existence?

Assessing the system's ability to resist these attack patterns requires knowledge about the system's internals and its development history. Did developers intentionally design the system to resist these types of attacks? Did developers take steps during the development process to avoid vulnerabilities that attackers might exploit? Was the system tested to assess its resistance to attacks, and, if so, how extensive was that testing? And how have changes to the system since it was first deployed affected the security of the implementation?

Consider the system's operational history. Has the system successfully demonstrated, in operation, its ability to resist these attack patterns, or has the system been successfully penetrated by attackers since it became operational, and what attack patterns did it succumb to?

Finally, estimate the severity of the risk. This can be a simple "high-medium-low" assessment, or maybe on a scale from one to ten. The estimating process is subjective and imprecise. The overall severity of risk is based upon the magnitude of the business risk involved, the likelihood of its occurrence, and the ability of the system to resist the attack. These intrinsically subjective factors are weighed against each other to reach an even more subjective conclusion.

The level of detail that this review will reach will largely depend on the business risks associated with the individual system. Most systems will present minimal business risks, and little effort should be committed to investigating the system's resistance to these minor problems, but systems that pose major business risks may require a more in-depth study to determine how well they can handle these more significant problems.

5. #dsy624-BSI_hoglund04

The result of this review is a list of the security risks posed by each system. Describe each security risk in terms of the business risk involved and a brief assessment of the system's ability to handle it, and note the severity estimate for the security risk.

Details are unimportant at this point. You want to know, at a management level, if the system poses a problem and some sense of the magnitude of that problem.

Step 4: Define Security Mitigation Projects

Describe how the enterprise will reduce the security risks associated with each legacy system to an acceptable level. Consider the circumstances associated with each system when developing these mitigation strategies. Recognize the advantages and pitfalls associated with the many mitigation options available [Zoufaly 02[6]].

Examine each legacy system individually. Review the risks#the potentially successful attack patterns#that the system faces. Get some sense of the changes to the system that would be necessary to make it resistant to these patterns.

Then consider your mitigation options. This decision often involves other considerations besides security. Is the current legacy system due for replacement anyway? Does it rely upon an obsolete technology? Is it a maintenance nightmare? Does it meet the current functional requirements of the users? Is its user interface lousy or obsolete? Is system documentation available? Is anyone still around who knows it well? Is it a resource hog?

These are your general options:

Option 1: Do nothing.

If the security risks represent a minor risk to the enterprise and the cost to address them cannot be justified, then note these conclusions but leave the system alone.

A corollary to Option 1 is to make some minor adjustments to the system's operational environment but not alter the system code itself. For example, you may need to establish some new policies or alter or clarify certain operational procedures. Maybe some training is required, or some tweaks to the network security appliances (firewalls, intrusion detection systems, etc.) are called for. Changes such as these may not actually eliminate the security risk but make it less likely to happen.

Option 2: Harden the legacy system.

If the security risks are significant and the effort (cost) to address them directly is reasonable and can be justified, then describe a project to modify the system to improve its ability to handle the risks. This option does not provide new functionality but simply eliminates vulnerabilities associated with the existing functionality.

A legacy system can be hardened in several ways. You can apply patches and corrections to the source code to eliminate coding bugs. With greater effort, you can alter the architecture of the system to avoid design flaws that attackers might exploit. Or you can integrate third-party COTS software into the system to prevent successful software exploits, using a "wrapper" approach to encapsulate and protect key functionality.

The decision to harden an existing legacy system, versus replacing it entirely, is often a tough one. You probably do not have a good estimate of the effort (cost) necessary to address the security issues. Any "harden the system" plan should begin with a task to develop a more solid, informed estimate of that effort. Then, based on the result of that initial task, you may want to reconsider the "harden the system" decision.

Another consideration with the "harden the system" decision is that fixes often introduce unintended problems. The "fix" is usually a patch intended to prevent something bad from happening. But these patches themselves can create new security vulnerabilities that attackers can exploit. Moreover, the patches can

6. #dsy624-BSI_zoufaly02

interfere with the legitimate operation of the system. Testing#regression and security#is very important when attempting to fix a security issue.

Option 3: Enhance the legacy system.

Consider making enhancements to the legacy system to address security concerns. These enhancements would replace or augment parts of the system that pose significant security concerns but retain intact other parts of the system that are otherwise performing properly and do not represent a security problem.

The enhancement option differs from the harden option in the extent of the modifications to the system. The changes would replace parts of the existing system completely.

The enhancement option differs from the replacement option because portions of the existing system will be retained unchanged. Also, the basic technology that the system is built upon is retained; a replacement decision would allow migration to a different technology.

The enhancement option is particularly attractive for large, complicated systems. Complete replacement of these systems would be very expensive and involve considerable business risks (project delays, budget overruns, etc.). Many systems include considerable background maintenance functionality that in itself operates satisfactorily and represents little security risk. But these systems often have other parts that face the public or are used widely within the enterprise and therefore represent the primary security concern. The enhancement option salvages the background functionality and focuses effort on replacing the other functionality that is of greatest concern.

Option 4: Replace the legacy system.

Replace the legacy system when the effort can be cost-justified. This is a major step that should not be undertaken lightly. Consider the business risks associated with any new software development effort. Realize the impact on users that moving to a new system will have.

Generally, system replacement cannot be justified based upon security concerns alone. The current system may be functionally obsolete or implemented with technologies that make it expensive to operate and maintain. A new system may provide business opportunities not currently available with the existing system.

Planning the transition to a new system is an important process. The security risk will continue unaddressed until the system’s replacement is available. New systems affect users; they will need training.

Implementing Legacy System Mitigation Projects

The assessment process described above identifies a series of projects to either harden or replace legacy systems that you consider to represent an unacceptable level of security risk to the enterprise. This section discusses these “fix” and “replace” projects in greater detail from a software security perspective.

Harden Legacy System Projects

A project to directly address security risks in an existing legacy system should follow the physician’s dictum, “do no harm”; you do not want to make the system worse than it was before you started.

Keep these points in mind:

Control Project Scope

Resist the “while we are in here, let’s fix that” impulse that occurs when you begin changing a system. Focus on the security risks that motivated the project in the first place.

This often is not possible in the real world. Management combines work to address security concerns with work to add new functionality or to correct known bugs. But management should understand the implications of doing this.

- The overall effort will be greater, and the elapsed time before the corrected system is in production will be longer.

- The work to fix the security problems will inevitably be commingled with the other work and may not receive the same level of attention as it would if the project was focused on security-only issues.
- Testing of the results of the security changes may be less intense because of the diversion of attention to functional testing issues.

Another common scope problem is fixing similar problems elsewhere in the system. You find a particular problem at one point that is causing the security risk you are trying to address, but you then realize this problem occurs elsewhere in the system. Should you only fix the one problem or fix it everywhere you can find it? The correct answer is not simple. Usually, all known security problems should be fixed, but that would significantly increase the scope of work, altering code in unexpected areas of the system. Before going forward with all the changes, consider the impact on your work effort, and be sure management is comfortable with this expansion of scope.

Understand the Security Risk

The people assigned to modify a legacy system must have the same understanding of the security issue that they are attempting to remedy as the analysts who identified the problem in the first place. Otherwise, they may correct the wrong problem.

Security issues can be terribly complicated and obscure. Moving from the business risk level of “if this happens, it’s a disaster” to the code level of function calls and “if” statements is a big jump. There is a lot of room for confusion and misinterpretation.

The security analysts who identify the security issues are responsible for clearly communicating them to the developers who will be working to address them. These represent the security requirements that the analysts expect the developers will satisfy.

An excellent way to express these security requirements is to describe misuse test cases that represent the problem [McGraw 06[7]]. A misuse test case describes how the system should react to a specific user action, intentional or unintentional, where the user submits inappropriate inputs. For example, when the program gets a certain input it should do this but never do that. Developers can use these test cases to guide their coding decisions and to define their testing protocols.

Security and Functional Testing

Efforts to fix security problems are notorious for creating new security vulnerabilities and for unintentionally affecting the legitimate operation of the system.

These undesirable side effects are common because fixes to security issues often add new code that is intended to prevent “bad” things from happening. The new code can have its own vulnerabilities, and often the definition of a “bad” thing is broad enough to interfere with the “good” things that the system is supposed to perform.

Write the new code while considering the advice offered in the other articles on the Build Security In web site. Carefully design this code to limit its effect on just the undesirable actions. However carefully you implement the changes, you must test the modified system extensively to build confidence that the correction has not caused new problems. First, perform tests to confirm that the fixes in fact did address the original problem. Also, conduct tests that attempt to compromise the system in other ways, to provide some assurance that the fix did not introduce some new security problems.

Subject the modified system to extensive functional testing as well as the security testing. The functional testing should uncover any disruptions in normal system usage caused by the insertion of the security correction code.

Ideally, developers will have previously created a series of regression tests intended to demonstrate the correct operation of the system; if they are available, use them. Nevertheless, usually you will need to develop some custom test cases. Design these functional tests to verify that the component that was modified is still operating properly.

---

7. #dsy624-BSI_mcgraw06

Enhance Legacy System Projects

Projects that address security concerns by replacing parts of a system's functionality face similar issues as projects that modify the existing functionality. The points about controlling scope and understanding the security risks at issue are equally valid in both situations. Testing is as always a major issue, but now new functionality created by the enhancement work must be subjected to both security and functional testing.

Enhancement projects are essentially development projects. They begin with a reassessment of the fundamental functional requirements of the system. They can involve radical design decisions that depart from the original system's design. Coding and testing of the new functionality is involved, and the integration of the new functionality with the existing functionality must be considered.

Much of the general commentary on the Build Security In web site applies to enhancement projects because they are development projects. Design and implementation decisions may be constrained somewhat because of the need to co-exist with the parts of the system being retained.

The major limitation of an enhancement project is that you must work within the limits of the technology on which the system was originally built. In some circumstances, building the new portions of the system in a different technology than was used for the system originally may be possible, but that decision would open up many new potential problems.

Replace Legacy System Projects

When you make the decision to replace an existing legacy system, you will follow the usual new system development process. This introduces all the problems and concerns of any system development effort.

All options are open. You can (and should) re-examine the functional requirements so the new system satisfies the present needs. You can implement the new system in a new technical environment, with a modern user interface.

Projects to replace legacy systems are not usually undertaken based solely on security issues. Enterprises replace systems when they do not support current business functions well, they are slow or difficult to use, or they are expensive to operate or maintain. These multiple motivations can complicate a replacement project.

Apply the Build Security In guidelines described elsewhere in this web site when implementing the new system. Define security requirements[8] as well as functional requirements. Conduct architectural risk analysis[9] on the design before implementation begins. During implementation, conduct secure code reviews[10] regularly, and test[11] components of the system for security flaws. Upon completion, perform penetration testing[12] based on predetermined threats and attack patterns[13].

Think of Security as a Process, Not a State

Systems change, and over the scope of a major enterprise, change is not a rare event, it is a day-to-day occurrence. Some changes are major and command your attention, while others occur below the radar. Some are business-driven while others are strictly technology-driven. Some are managed by the IT organization, others by users (within or outside accepted policy), and others happen without notice as Internet-connected software is automatically updated.

Beyond the changes in the actual systems, the computing world changes#the practices and tools of the black hat community are continually evolving. Laws and regulations also change, perhaps making what was once a minor risk a major one.

8. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements.html (Requirements Engineering)
9. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/architecture.html (Architectural Risk Analysis)
10. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/code.html (Code Analysis)
11. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/testing.html (Security Testing)
12. http://buildsecurityin.us-cert.gov/bsi/articles/best-practices/penetration.html (Penetration Testing)
13. http://buildsecurityin.us-cert.gov/bsi/articles/knowledge/attack.html (Attack Patterns)

Many changes, perhaps most, will not affect security. Others will. Sometimes it can be difficult to tell the difference. In any case, just making a system secure once (if that is actually possible) is not enough.

Entropy is irresistible. The order you achieve through the Herculean remediation effort will degrade as the systems evolve. You must develop a continuing program of securitization. Consider security in all system changes. Establish change control systems to monitor system changes, and, periodically, audit these systems to find out what you missed.

Conclusion

Your inventory of existing legacy systems represents a potential security risk to your enterprise. You can gain a deeper understanding of the magnitude of that risk and develop a strategy for reducing it to acceptable levels by applying the steps described above.

Glossary

security vulnerability	A design flaw or code bug that an attacker could exploit to compromise a system.
security risk	A potential threat to an enterprise represented by the exploitation of a security vulnerability, generally proportional to the likelihood the vulnerability will be exploited and the impact on the enterprise if it is.
legacy system	Business application system currently in use within the enterprise.
mainframe	Large-scale data processing equipment intended to support bulk data processing tasks.
minicomputer	Medium scale data processing equipment generally designed to support real-time processing tasks, including providing online interactive support for user terminals.
terminal	A device for entering information into an attached computer and displaying information from that computer, with no business logic manipulation of that information.
workstation	A computer capable of executing business logic that interacts with other computers to perform data processing tasks.
server	A computer dedicated to providing a particular service (often database or file sharing) to attached workstations.
client/server	A particular data processing architecture in which a workstation executes business logic that interacts with information stored remotely in a centralized repository.

References

[Glass 03]	Glass, Robert L. *Facts and Fallacies of Software Engineering*. Boston, MA: Addison-Wesley, 2003.

[Hoglund 04]	Hoglund, Greg & McGraw, Gary. *Exploiting Software: How to Break Code*. Boston, MA: Addison-Wesley Professional, 2004 (ISBN 0-201-78695-8).
[McGraw 06]	McGraw, Gary. *Software Security: Building Security In*. Boston, MA: Addison-Wesley Professional, 2006 (ISBN 0-321-35670-5).
[Zoufaly 02]	Zoufaly, Federico. "Issues and Challenges Facing Legacy Systems[14]." developer.com, November 1, 2002.

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com